UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

SISTEMA DE VISUALIZACIÓN DE INFORMACIÓN DE SENSORES PARA LA AUTORREGULACIÓN EMOCIONAL

Lucía Rodríguez González
Tutor: Javier Gómez Escribano
Ponente (si procede): Germán Montoro Manrique

Mayo 2017

SISTEMA DE VISUALIZACIÓN DE INFORMACIÓN DE SENSORES PARA LA AUTORREGULACIÓN EMOCIONAL

AUTOR: LUCÍA RODRÍGUEZ GONZÁLEZ
TUTOR: JAVIER GÓMEZ ESCRIBANO

Escuela Politécnica Superior
Universidad Autónoma de Madrid
MAYO 2017

Resumen (castellano)

Este Trabajo de Fin de Grado ha sido desarrollado con el objetivo de manejar la gran cantidad de datos que pueden recoger los relojes inteligentes (smartwatches), con un fin específico: ayudar al entendimiento del comportamiento de las personas con Trastornos del Espectro Autista (TEA), ya que muchas veces su conducta es imprevisible.

Para ello, se ha utilizado la aplicación **Taimun-Watch** [1], desarrollada por el laboratorio Ambient Intelligence Laboratory (AmILab) de la EPS [2], en colaboración con el Instituto de Psico-Pediatría Quintero Lumbreras [3]. Este sistema monitoriza las constantes del usuario (con TEA) mediante un smartwatch y, en base a las medidas recogidas, detecta situaciones de estrés. En caso de darse una de éstas, el reloj muestra una serie de estrategias de regulación, que previamente ha programado la persona de apoyo[1], para ayudarle a calmarse.

Gracias a la recolección de los datos que registra el reloj en la aplicación, que está monitorizando al usuario todo el tiempo que lo lleva puesto, es posible llevar a cabo una representación gráfica de los mismos, con el fin de estudiar patrones de comportamiento e indicadores que ayuden a las personas de apoyo de estos usuarios a entender qué sucede a lo largo de su día a día.

Para lograr este objetivo, se ha desarrollado, en Android, una sección extra de la aplicación Taimun-Watch, de representación de estadísticas, dividiendo las mismas en estadísticas básicas y avanzadas, manteniendo el aspecto original de la aplicación, e intentando que sea lo más clara y sencilla posible, pues está destinada a un tipo de usuario con conocimientos básicos en aplicaciones Android.

El resultado final de este proyecto ha sido satisfactorio y completamente funcional, y actualmente el proyecto está operativo, ejecutándose correctamente, en la aplicación Taimun-Watch.

Palabras clave (castellano)

Trastorno del Espectro Autista, TEA, Taimun-watch, representación de gráficas, estadísticas, Android, smartwatch

[1] Padres, tutores o profesores que se encargan de ayudar a las personas con TEA.

Abstract (English)

This Bachelor Thesis has been developed for handling the large amount of data that, nowadays, smartwatches can collect, with a specific goal: to help understanding the behaviour of people with Autism Spectrum Disorders (ASD), as many times, their conduct is unpredictable.

To achieve this aim, the application **Taimun-Watch** [1], developed in the Ambient Intelligence Laboratory (AmILab), at the EPS [2], collaborating with the Instituto de Psico-Pediatría Quintero Lumbreras [3], has been used. This system monitors the beats of the user (with ASD) by using a smartwatch, and considering the measures taken by the watch, the application can detect stress situations. If this happens, the watch shows a number of regulation strategies, previously programmed by the support person[2], in order to calm him down.

Thanks to the data compilation that the smartwatch can record in the application, monitoring the user all the time he wears it, it is possible to make a graphic representation of this data, in order to study behaviour patterns and markers that help support people of these users to understand what happens throughout their day to day.

To achieve this goal, a new section of the application Taimun-Watch has been developed, in Android language, for statistic representation, by dividing them into basic and advanced statistics, keeping the original appearance of the application and trying to be as clear and simple as possible, as it is aimed at a specific type of user with basic knowledge in Android applications.

The final result of this project has been satisfactory and fully functional, and the project is currently operational as well as running correctly, included in the Taimun-Watch application.

Keywords (inglés)

Autism Spectrum Disorders, ASD, Taimun-watch, graphic representation, statistics Android, smartwatch

[2] Parents, tutors or teachers who are in charge of helping people with ASD.

Agradecimientos

A mis padres, mis hermanos, mis tíos, mis primos, mis abuelos y mis amigos, por apoyarme, no solo con este Trabajo de Fin de Grado, sino durante toda la carrera, y darme ánimos para conseguir todo lo que me proponga.

A mi tutor, Javier Gómez Escribano, por brindarme la oportunidad de realizar este proyecto y dedicarme tanto tiempo, y a todos los miembros de AmILab, que me han ayudado siempre que he tenido algún problema con el desarrollo del proyecto.

Gracias también al IPP Quintero Lumbreras por hacer posible este proyecto y por toda la ayuda e ideas que han aportado al mismo.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Entre el amplio abanico de posibilidades que abren ante nosotros las nuevas tecnologías, se haya el camino de la adaptación de los dispositivos móviles a las necesidades de los que más lo necesitan, en este caso, personas con Trastornos del Espectro Autista (TEA).

El proyecto que se describe en este TFG, pretende facilitar el seguimiento del estado emocional de dichas personas, gracias a la recopilación de datos y a la gestión de momentos de crisis de la aplicación móvil "Taimun Watch", cuyo objetivo es ayudar a la autorregulación emocional de las personas con TEA, mediante una serie de regulaciones y estrategias que la persona de apoyo ha programado previamente para que actúen cuando se detecte un momento de estrés del usuario con TEA.

La idea surge, de querer aprovechar los beneficios que nos brindan los relojes inteligentes, al ser capaces de recoger una serie de medidas mediante los sensores con los que cuentan. Con estas medidas, somos capaces de visualizar una pauta de comportamiento que, en el caso de las personas con TEA, puede ser extremadamente útil a la hora de determinar qué les hace sentir más cómodos o qué les lleva a una situación de estrés, ya que, en muchas ocasiones, no son capaces de expresar claramente cómo se sienten en cada momento.

La aplicación móvil anteriormente mencionada ya cuenta con la recolección de estos datos, por lo que mi objetivo se ha centrado, en la representación de los mismos, de una manera que puedan entender las personas que están en contacto directo con los usuarios de la aplicación: las personas de apoyo. Por este motivo, ha sido adaptada para que sea comprensible por cualquier usuario con conocimientos básicos de aplicaciones móviles.

1.2 Objetivos

El objetivo de este TFG es el planteamiento, desarrollo e implementación de un sistema de representación gráfica y visual de la información de los sensores de un reloj inteligente, en la aplicación Taimun-Watch, mostrando los datos de forma básica y de forma avanzada.

En la forma básica, se detallarán los momentos de crisis del usuario del reloj y los datos clave de cada evento, permitiendo acceder a la edición de la regulación de dicho evento al pulsar sobre él.

Para la visualización de los datos de forma avanzada, se podrán comprobar todos los parámetros que nos muestran los distintos sensores disponibles, los cuales se podrán elegir una vez se esté en la gráfica avanzada, así como se pueden seleccionar los ejes que se desean visualizar, siempre que el sensor elegido tenga más de un eje.

La visualización de estas estadísticas debe seguir la misma pauta visual que el resto de la aplicación a la que pertenece esta sección, con un diseño sencillo e intuitivo.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1.** Introducción: motivación y objetivos del proyecto.
- **Capítulo 2.** Estado del arte: librerías de representación de gráficas analizadas y comparadas.
- **Capítulo 3.** Diseño: análisis y metodología aplicada.
- **Capítulo 4.** Desarrollo: lógica de la aplicación e implementación.
- **Capítulo 5.** Integración, pruebas y resultados.
- **Capítulo 6.** Conclusiones y trabajo futuro.

2 Estado del arte

En este capítulo, se trata el estudio de las diferentes librerías de representación de gráficas para Android que existen actualmente y que podemos usar en el proyecto, comparando sus características y los pros y contras que nos interesan en particular. Al final del capítulo se puede ver una tabla que resume las características más importantes de cada una.

2.1 Librerías de representación de gráficas para Android

A la hora de empezar con el desarrollo de la aplicación, comienza la búsqueda de la mejor forma de representación gráfica, ya que, en lugar de tener que desarrollar una librería de gráficas para Android, podemos elegir una de las muchas que hay en Internet, con código abierto, y adaptarla para usarla en el proyecto.

Después de una primera selección entre todas las librerías con código abierto que pudimos encontrar, elegimos cuatro que destacan por sus características y que procedemos a comparar. Estas son: *WilliamChart*, *AndroidPlot*, *MPAndroidChart* y *GraphView*.

2.1.1 Androidplot

En primer lugar, analizamos Androidplot [4], que se vende a sí misma como "una librería para crear gráficos dinámicos y estáticos en aplicaciones Android", que además cuenta con gran compatibilidad ya que funciona desde Android 1.6 en adelante y, en teoría, se usa en más de 1000 aplicaciones de Google Play. A decir verdad, el diseño es lo suficientemente simple como para que puedan usarlo gran cantidad de dispositivos, aunque visualmente no es muy atractivo.

Con esta librería se realizaron una serie de pruebas iniciales, usando datos aleatorios, para tener una primera idea del funcionamiento, pero nos encontramos con una implementación bastante compleja y poco intuitiva.

Además, se pudo comprobar que esta librería no funcionaba de manera fluida al representar un número elevado de datos, y ya que nosotros necesitamos poder representar un número mínimo de datos bastante grande, decidimos probar otras librerías.

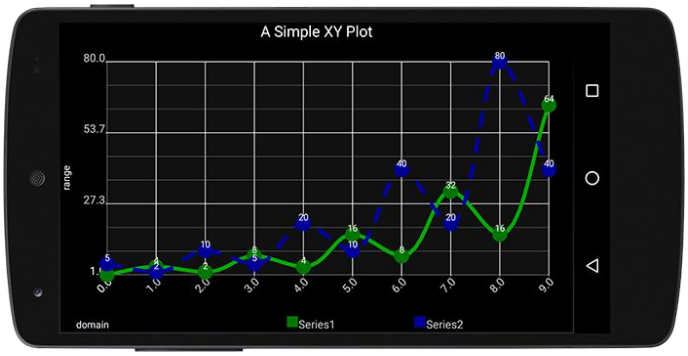

Figura 2-1: Ejemplo de una gráfica simple con Androidplot

2.1.2 WilliamChart

Con una interfaz mucho más atractiva que la anterior, WilliamChart [5] pretende ser una librería de representación de gráficas intuitivas y visualmente sencillas, con gráficos simples y limpios.

También cuenta con la posibilidad de mostrar las gráficas en un dispositivo weareable, introducir animaciones y personalizar cada gráfica.

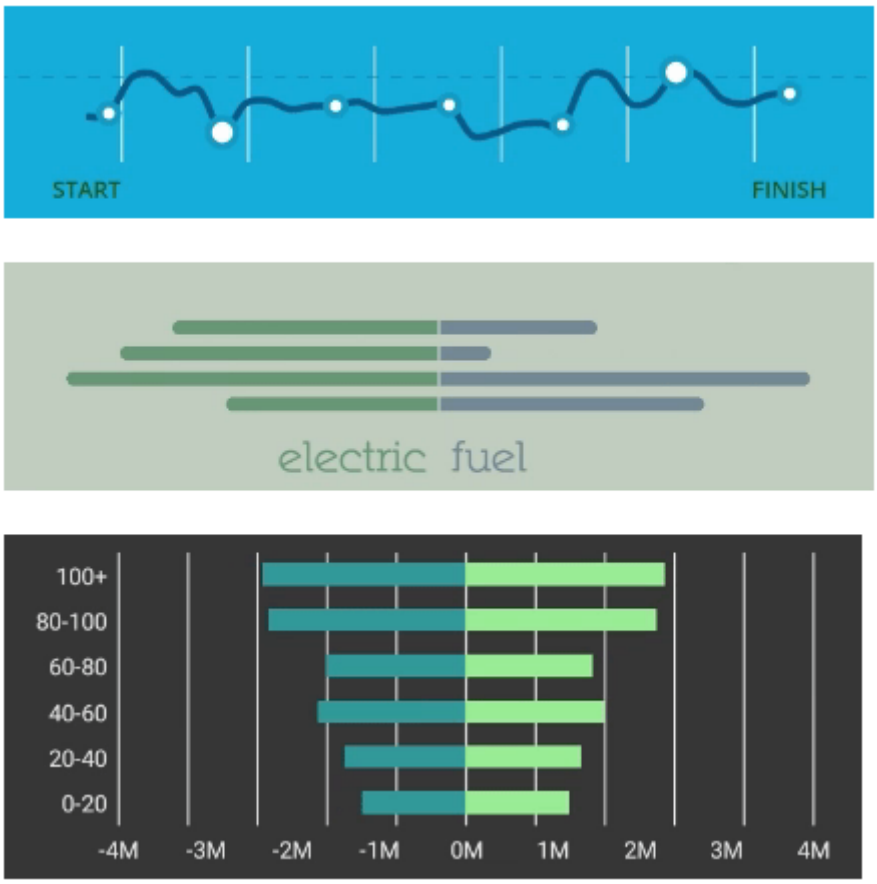

Figura 2-2: Ejemplo de posibles gráficas con WilliamChart

Aunque este diseño minimalista tiene un buen aspecto a primera vista, al analizarlo detenidamente nos damos cuenta de que es, de hecho, demasiado sencillo, pensado para pequeños datos, con marcas de una vez al mes, o de, como mucho, un dato al día, con muchos diseños distintos pero que, para nuestros datos, no es viable, ya que contamos con miles de datos en un solo día.

Además de esto, comprobamos también que sus gráficas, al ser tan simples, no cuentan con la posibilidad de hacer zoom sobre una zona, pues la cantidad de datos para los que están pensadas estas representaciones se ven a simple vista. El zoom es un elemento importante en la vista de nuestras gráficas, ya que nos permite visualizar los datos en su totalidad o los de algún intervalo concreto, por lo tanto, esta carencia influye también en la elección de la librería adecuada.

2.1.3 MPAndroidChart

Con una disponibilidad de uso desde el nivel 8 de API en adelante y muchas posibilidades distintas de representación de gráficas, MPAndroidChart [6] parecía, en un primer momento, una buena opción para el proyecto. También cuenta con la posibilidad de representar gran cantidad de datos y hacer zoom sobre una zona, así como varias opciones distintas de visualización de la gráfica como quitar el relleno, los puntos de los ejes… que, en resumen, resultan ser demasiadas cosas que no necesitamos.

Al contrario que en el caso de las otras librerías, que tenían algunas carencias para lo que nosotros buscábamos, esta librería tiene demasiadas opciones sobrantes, que, a nivel de código, son más o menos difíciles de eliminar de la gráfica dependiendo del caso, pero, aun así, pudiendo encontrar otra librería que cumpla mejor con nuestras necesidades, preferimos dejar esta aparte.

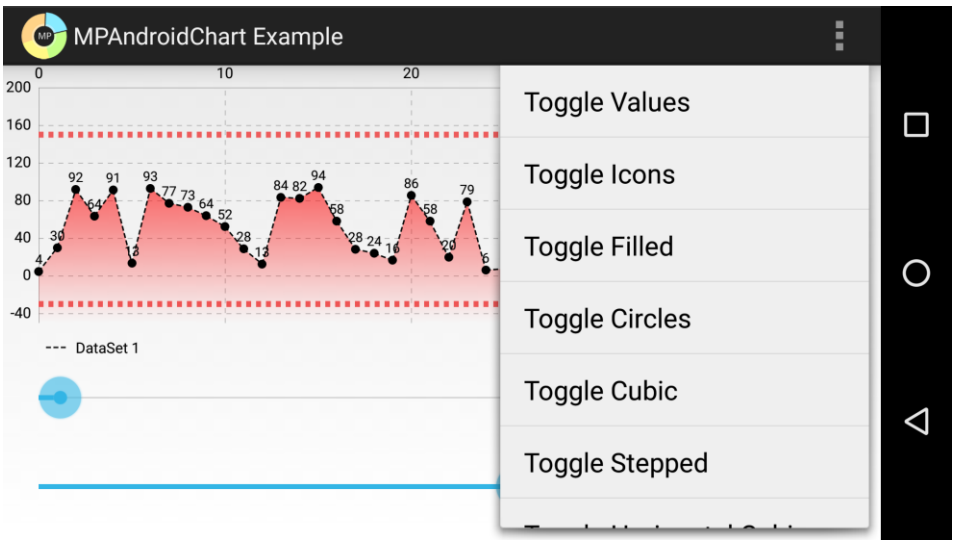

Figura 2-3: Ejemplo del gráfico simple con diferentes opciones de visualización de la librería MPAndroidChart

Esta librería, además, cuenta con una opción para la representación de un gran número de datos (en teoría hasta 30.000), que es algo que en un principio podría interesarnos, ya que las otras librerías están muy limitadas en este aspecto, pero a la hora de probarlo, nos dimos cuenta de que, realmente no es necesario llegar a este nivel de representación, pues mostrar tantísimos datos dificulta la lectura de la gráfica y no aporta tanta información como puede parecer en un primer momento.

Por lo tanto, podemos concluir que está bien poder representar más datos que los que nos permiten las otras librerías, pero con un límite de puntos máximos.

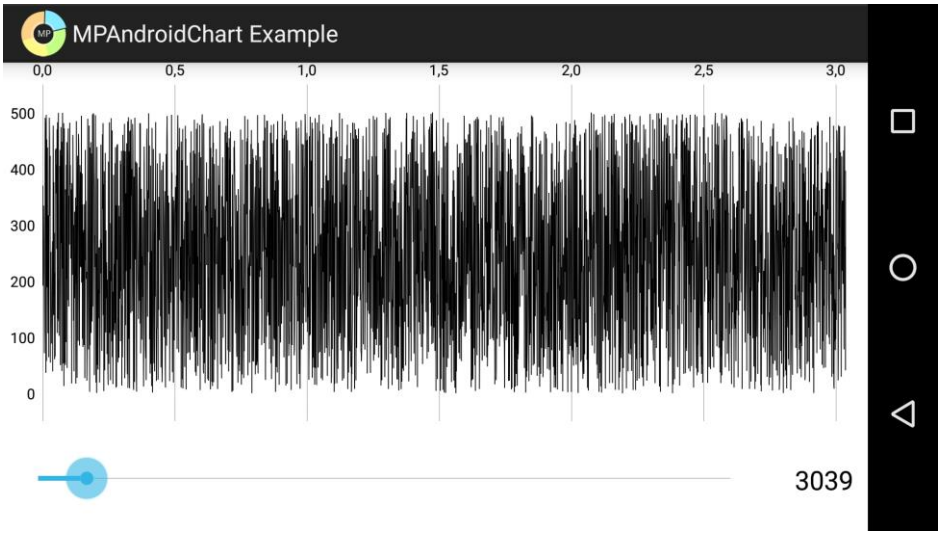

Figura 2-4: Gráfica representada mediante MPAndroidChart de una gran cantidad de datos y variación

2.1.4 GraphView

Por último, la opción elegida para nuestro proyecto fue esta librería, GraphView [7], cuyas características principales se detallan a continuación.

En primer lugar, es una librería que cuenta con una gran variedad de tipos de gráficas entre las que elegir, de las que nosotros elegimos el modelo simple con opción de hacer scroll sobre el eje X y zoom sobre una zona cualquiera, lo que cubre a la perfección nuestras necesidades en el aspecto visual.

Además, en cuanto a la codificación, cargar los puntos en la gráfica es mucho más sencillo que con las otras librerías mencionadas anteriormente, y esta cuenta con una aplicación de demo en la que podemos ver código de demostración para guiarnos en el desempeño de nuestros requisitos, lo cual no poseían todas las demás. Para cargar estos puntos únicamente hay que crear un tipo de dato "DataPoint(x,y)" por cada punto que queramos representar, añadirlo a una o más "Serie" y añadir estas al "Graph".

Esta librería cuenta, por supuesto, con la libre personalización de los detalles de las gráficas, como las leyendas, los nombres de ejes, el estilo de representación, etc.

Como extra, también cabe mencionar, que esta librería brinda la opción de crear puntos de datos con fechas, es decir, dando el *timestamp* es capaz de convertirlo a la hora o fecha correspondiente en formato legible, lo cual las otras librerías no contemplaban. Sin esta opción adicional, nos habríamos visto obligados a desarrollar un conversor que mostrase los datos de tiempos en forma de fecha en la gráfica, mientras que internamente se manejasen en formato *timestamp*, ya que nuestros datos siempre están compuestos por, al menos, un eje de fechas, que necesariamente hay que convertir para su lectura.

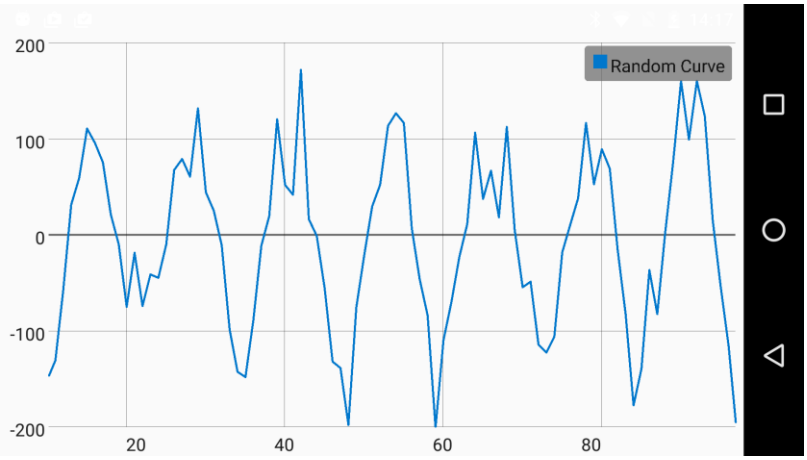

Figura 2-5: Ejemplo de una gráfica simple con GraphView

2.2 Tabla comparativa de las librerías

	Androidplot	**WilliamChart**	**MPAndroidChart**	**GraphView**
Compatibilidad	Android 1.6 en adelante (API 4)	Android 4 en adelante (API 15)	Android 2.2 en adelante (API 8)	Android 2.3 en adelante (API 9)
Tipo de diseño	Poco intuitivo y poco atractivo	Muy intuitivo, simple y atractivo	Muy configurable pero muy poco intuitivo y muy complejo	Muy intuitivo y simple
Coste de implementación	Bastante complejo para el nivel que necesitamos	Coste alto, sin ejemplos de implementación	Coste medio	Coste medio, con ejemplos de implementación
Manejo de grandes cantidades de datos	Muy lento	No es posible	Hasta 30.000 datos en la misma gráfica	Hasta unos 3.000 datos en la misma gráfica con fluidez
Zoom	Sí, lento con nivel medio de datos	No	Sí	Sí

Tabla 2-1: Comparativa de librerías

3 Diseño

A continuación, se desarrolla el análisis de requisitos del proyecto, a partir del cual hemos elegido una metodología que se adecúa a nuestras necesidades explicando el porqué de su elección.

3.1 Análisis de requisitos

Para lograr el objetivo del proyecto, se han acordado los siguientes requisitos funcionales y no funcionales.

3.1.1 Requisitos funcionales

RF1 – Selección entre vista básica o avanzada

La aplicación permitirá al usuario cambiar entre estos dos tipos de vista para los datos de cada día. Para ello, tras seleccionar el día a mostrar, aparecerá la lista de eventos de la vista básica y desde esta pantalla habrá otro botón para acceder a la vista avanzada, desde la cual también se puede volver a la vista básica de nuevo.

RF2 – Filtro de fechas

Desde la pantalla principal de las estadísticas, se podrá ver una lista de fechas que tienen registros, las cuales pueden filtrarse por fecha de inicio y de fin para acotar la lista de tarjetas.

RF3 – Acceso a edición de regulación desde la vista básica

Con solo pulsar el evento requerido, se accederá a la edición de la regulación que actuó cuando se produjo ese evento, indicada en la tarjeta seleccionada.

RF4 – El modo básico muestra el nombre de la regulación, la hora de inicio y fin del evento y las pulsaciones en esos momentos

Las tarjetas de la vista básica indicarán el nombre de la regulación que se activó en el momento de la crisis, la hora a la que comenzó, las pulsaciones que marcaba el reloj en ese momento y así mismo la hora de fin del evento y sus respectivas pulsaciones.

RF5 – El modo avanzado permite elegir el fichero a mostrar

Desde la pantalla de gráfica avanzada, se podrá elegir mediante un botón qué fichero de los disponibles del día solicitado se desea representar.

RF6 – Modo avanzado permite elegir los ejes a mostrar

Al seleccionar el fichero del cual queremos mostrar su vista avanzada, se podrá elegir el eje o los ejes que se quieran mostrar en ese momento, siempre que el fichero en cuestión cuente con más de un eje.

RF7 – El modo avanzado podrá representar cualquier fichero que cumpla con el formato indicado

Siempre que se cumplan los requisitos de formato se podrán representar los ficheros en el modo avanzado. Este formato necesita:

- Título: nombre de los datos sin espacios, "- ", fecha separada por "_"
- Formato: .txt

Ejemplo título: acelerometro-15_4_2017.txt

- Interior del fichero:
  - Cada columna deberá estar separada por una tabulación
  - Primera fila con las cabeceras de las columnas (títulos y nombres de ejes)
  - La primera columna siempre tendrá el *timestamp*
  - La segunda columna siempre tendrá la precisión (accuracy)
  - La tercera columna será el primer eje de la gráfica
  - Desde la cuarta columna en adelante serán ejes optativos

3.1.2 Requisitos no funcionales

Usabilidad

RNF1 – El tiempo medio de aprendizaje de uso de la aplicación será no superior a 10 minutos.

RNF2 – En casos de carga de datos se mostrará un indicador informando de la misma.

Accesibilidad

RNF3 – Los mensajes y textos de la aplicación estarán disponibles en inglés y castellano.

Interfaz

RNF4 – La interfaz siempre será lo más simple e intuitiva posible, sin sobrecargar la vista, y siguiendo la estética y el diseño del resto de la aplicación.

Plataforma

RNF5 – La aplicación solo es compatible con los dispositivos Android.

RNF6 – Plataforma soportable desde API 20 en adelante.

Espacio

RNF7 – El espacio necesario para la instalación de la aplicación es de 8MB, más el espacio adicional variable de los ficheros de datos.

Permisos

RNF8 – La aplicación necesita contar con permisos de acceso a los archivos del dispositivo.

3.2 Metodología

Para el desarrollo del sistema de visualización, se ha llevado a cabo un diseño centrado en el usuario incremental iterativo. Centrado en el usuario porque el objetivo del proyecto está condicionado por el uso que van a hacer de él las personas a las que va dirigido: las personas de apoyo de los usuarios de la aplicación Taimun-Watch, por lo que nos centramos en objetivos de usabilidad y accesibilidad como características prioritarias, teniendo siempre en cuenta el RNF4; incremental, porque así podemos desarrollar el producto por partes e irlas integrando a medida que se van completando, sin necesidad de tenerlo todo a la vez; y diseño iterativo, porque tras cada iteración se revisa y mejora el producto.

El diseño centrado en el usuario tiene por objetivo la creación de productos que resuelvan las necesidades concretas de los usuarios finales, consiguiendo la mayor satisfacción y mejor experiencia de uso posible con el mínimo esfuerzo por su parte, por ello, cada decisión tomada en el diseño del proyecto, tiene como objetivo principal el aseguramiento de la usabilidad del sistema, basándonos en las necesidades, objetivos, expectativas, motivaciones y capacidades de los usuarios [8].

Por otra parte, como podemos ver de manera resumida en la figura 3-1, el modelo incremental iterativo nos permite ir desarrollando el proyecto poco a poco, añadiendo funcionalidades nuevas en cada iteración. De esta forma, partimos de una versión básica del proyecto con la que podemos experimentar y sacar nuevas ideas o implementaciones, o eliminar secciones que no nos interesan, y volver a hacer otra iteración con estos cambios, volverlo a probar y volverlo a actualizar. Tras cada iteración se lleva a cabo una reunión con el tutor y expertos del Instituto de Psico-Pediatría Quintero Lumbreras, con el que participamos en el proyecto, en la que decidimos qué cosas se podrían cambiar, añadir, o eliminar de la versión actual.

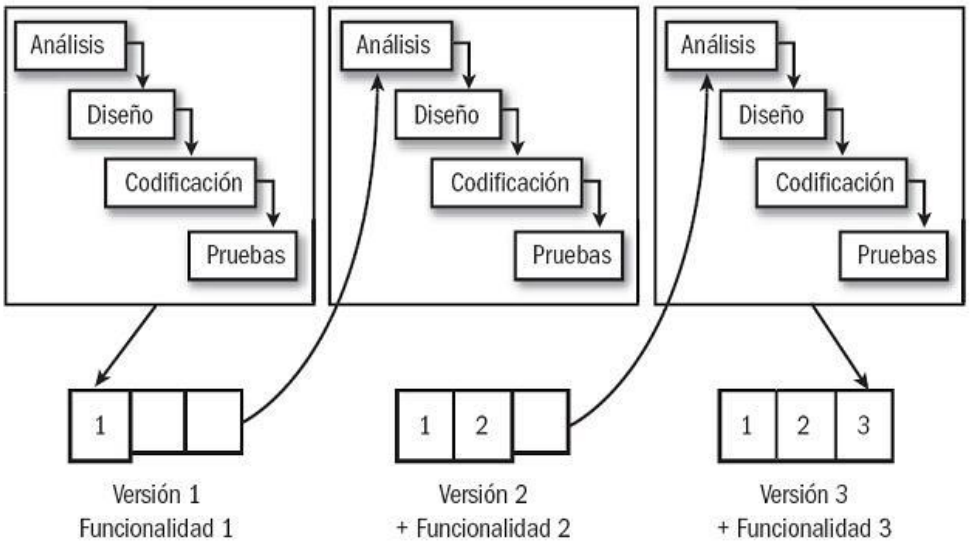

Figura 3-1: Resumen del modelo incremental iterativo [9]

En el caso de este proyecto, se realizaron 3 iteraciones que marcaron el desarrollo del mismo.

Primera iteración
En la primera reunión del proyecto, se decidió que habría una vista principal en la que poder elegir la fecha de los datos a visualizar, y que desde ahí se llegase a una sola gráfica que representase todos los datos del día seleccionado, pero al hacerlo de esta manera nos dimos cuenta de que el usuario podría necesitar ver los datos de una forma más simple y resumida que con una gráfica que lo mostrase todo a la vez.

Segunda iteración
Decidimos mantener la selección de fecha de los datos a mostrar, pero dando la opción de elegir una vista básica o una vista avanzada, donde la vista básica contendría algún tipo de formato visual en el que se distinguiese claramente dónde sucedían los puntos de crisis (momentos en los que las pulsaciones del usuario de la aplicación superaron el umbral establecido) y qué regulación había saltado en cada caso.

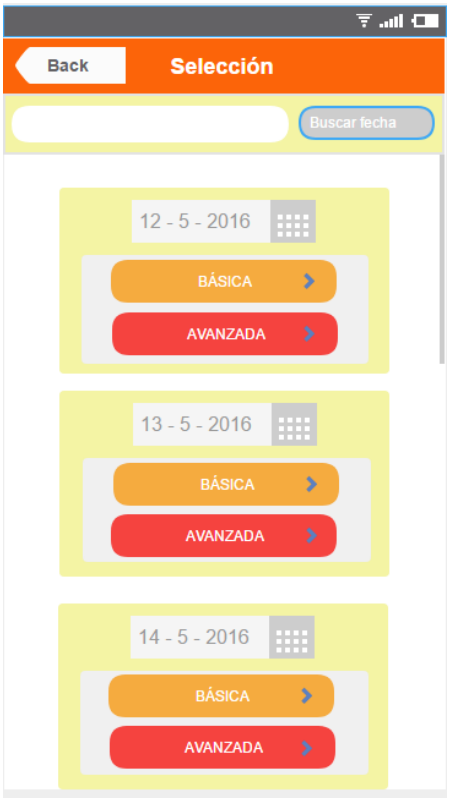

Figura 3-2: Maqueta de selección de datos

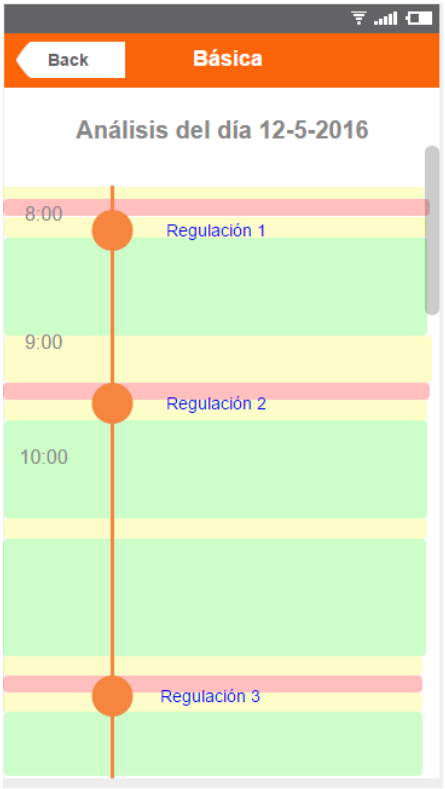

Figura 3-3: Maqueta de vista básica 1

Y donde la vista avanzada diese a elegir entre los distintos ficheros de datos a representar y los ejes a visualizar en el caso de que la gráfica del fichero seleccionado tuviese más de un eje (figura 3-4).

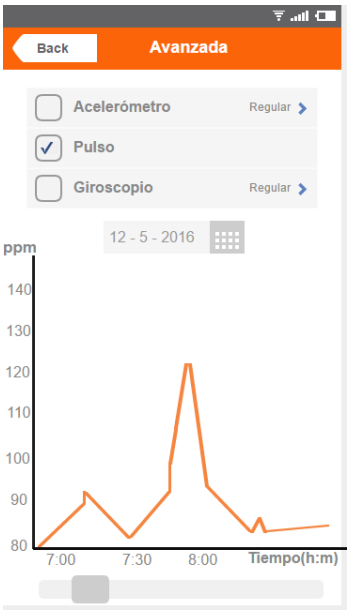

Figura 3-4: Maqueta de vista avanzada 1

Tercera iteración

Finalmente, y tras probar el diseño de la segunda iteración, decidimos que, en lugar de seleccionar el tipo de vista desde la pantalla de selector de fechas, era más efectivo que al pulsar sobre la fecha elegida se navegase directamente a la vista básica y que desde ahí se pudiese ir a la vista avanzada, ya que el usuario general de la aplicación casi siempre visualiza únicamente las estadísticas básicas, y deja las avanzadas para casos puntuales en los que quiera comprobar algún parámetro concreto.

Además, se decidió que la gráfica a mostrar por defecto al entrar en la vista avanzada fuese la del pulsómetro, ya que son los datos de las pulsaciones los que definen la actuación de las regulaciones, de la otra parte de la aplicación, cuando se producen los momentos de crisis.

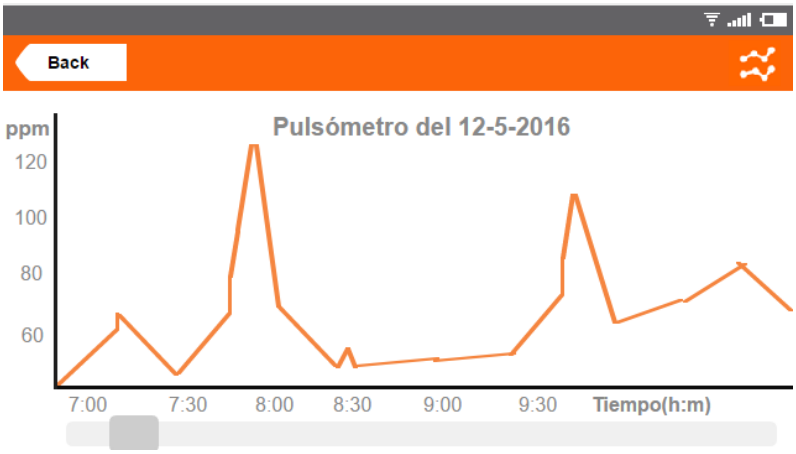

Figura 3-5: Maqueta de vista avanzada 2

También se cambió el aspecto de la vista básica del planteado inicialmente, para que conservase la apariencia general del resto de la aplicación y para hacerlo lo más sencillo y legible posible, usando el mismo diseño de tarjetas que se usa en otras secciones de la aplicación, con la información más importante del evento registrado en cada caso (figura 3-6) como se detalla en el Requisito Funcional 4 (RF4).

Figura 3-6: Maqueta de vista básica 2

4 Desarrollo

La explicación del desarrollo y la implementación llevada a cabo en el proyecto, comienza con la explicación de la obtención de los datos y la estructura de los ficheros en los que están recogidos, para el posterior acceso a los mismos. Seguidamente, se distingue entre el desarrollo de la visualización de estadísticas básicas y de estadísticas avanzadas.

4.1 Obtención de los datos

Para llevar a cabo la representación de los datos, primero hay que obtenerlos. Esto se consigue gracias a la acción del reloj inteligente o smartwatch, que recoge los datos que los sensores proporcionan sobre el usuario que lo lleva puesto, en nuestro caso, la persona con TEA, y gracias al sistema de recopilación de datos de la aplicación Taimun-Watch, al sincronizar reloj y teléfono inteligente, los datos son almacenados en forma de ficheros en la ruta de la aplicación en el teléfono.

Una vez almacenados, se puede acceder a ellos para proceder a su lectura y representación, gracias a lo cual, las personas de apoyo, podrán evaluar el comportamiento del usuario a lo largo de cada día, analizando las estadísticas y gráficas.

En la siguiente figura se puede ver un esquema de los pasos que se siguen desde la obtención de los datos hasta su interpretación.

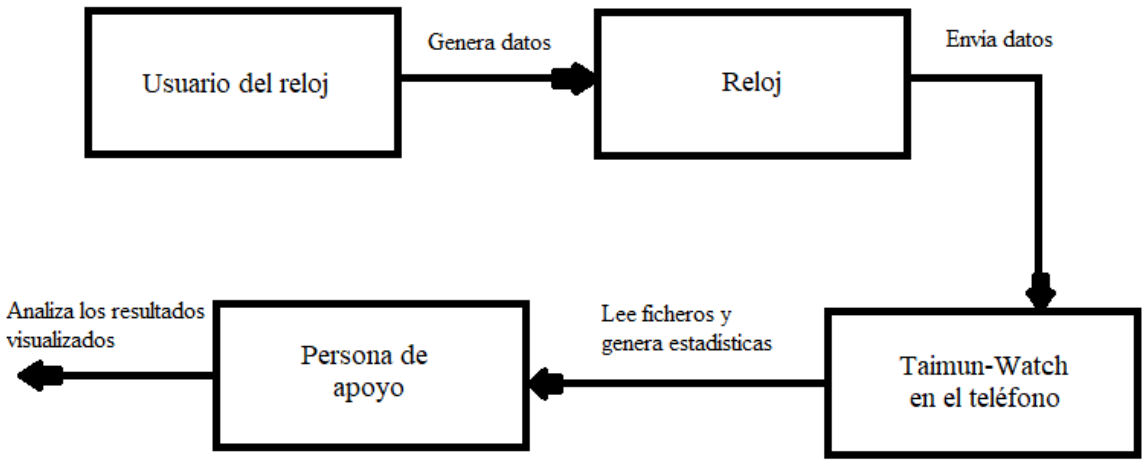

Figura 4-1: Esquema de la obtención y procesado de datos

4.2 Estructura y manejo de los ficheros de datos

Para llevar a cabo ambas representaciones, básica y avanzada, se accede a los ficheros que ya han sido guardados en el teléfono, almacenados en las rutas: *Carpeta de la aplicación*/files/logs/*id-del-reloj*/**events** y *Carpeta de la aplicación*/files/logs/*id-del-reloj*/**sensors.**

En *events*, se guarda un fichero por cada día con datos, con la fecha de ese día en el título, en el que quedan registrados el *timestamp* y el evento (columna *event*) que sucede en ese momento (con las pulsaciones de ese instante en determinados eventos).

Estos eventos pueden indicar: una evaluación negativa cuando el ritmo cardíaco es normal, una evaluación positiva en la que se inicia una regulación, las estrategias de esa regulación que se muestran en la pantalla del reloj, la finalización de la regulación, o los pasos con las imágenes que se han mostrado en el reloj para cada estrategia:

1493798695235	EVALUACION_POSITIVA 87.11111111111111
1493798695292	INICIO_REGULACION Saltar y sentarse 2
1493798695317	INICIO_ESTRATEGIA_Saltar
1493798695317	INICIO_PASO_2130837759.png
1493798710791	INICIO_ESTRATEGIA_Sentarse
1493798710791	INICIO_PASO_2130837773.png
1493798716051	FIN_REGULACION
1493800981655	EVALUACION_NEGATIVA 81.83333333333333
1493801011688	EVALUACION_NEGATIVA 78.75
1493801041864	EVALUACION_NEGATIVA 75.28571428571429

13/68

Figura 4-2: Ejemplo de un fichero *events*

Por otro lado, la carpeta *sensors* recoge la información de todos los sensores observados de cada día con datos, por lo que, de un mismo día, tendremos tantos ficheros diferenciados como sensores estemos tratando. Así, si estamos procesando la información de los sensores de acelerómetro, giroscopio, pasos y pulsómetro como es nuestro caso, tendremos 4 ficheros distintos por cada día con datos.

La composición de estos ficheros ya ha sido explicada anteriormente en el RF7.

timestamp	Accuracy	x	y	z
1493798394901	2	8.455688	-0.42674255	4.8788757
1493798395155	2	8.43367	-0.4625702	4.822464
1493798395286	2	8.385941	-0.36331177	5.143692
1493798395475	2	8.465088	-0.3565216	5.3285217
1493798395670	2	8.595886	-0.28605652	4.893875
1493798395868	2	8.343567	-0.33033752	4.926605
1493798396066	2	8.288345	-0.37806702	5.714264
1493798396264	2	8.820358	-0.3597412	4.26915
1493798396461	2	8.252518	-0.17001343	5.144882

1/30+

Figura 4-3: Ejemplo de un fichero de acelerómetro (3 ejes) en *sensors*

4.3 Agrupamiento por fechas

El primer paso para mostrar la lista de los días con datos disponibles, es agrupar los ficheros que encontramos en la carpeta de sensores por fechas, para tener una sola tarjeta seleccionable por fecha, y de esa agrupación guardamos las rutas a los ficheros que contienen y la ruta al fichero de eventos de ese día, para así, cuando se pulse una de esas tarjetas poder mostrar la vista básica usando esas rutas.

Las tarjetas llevan asociado un *RecyclerView* de manera que no se cargan todas las tarjetas a la vez en la vista, sino que solo lo hacen las que se ven en ese momento, lo que supone un consumo de recursos menor y más eficaz.

Además, para cumplir con el RF2, esta vista cuenta con un filtro para mostrar solo las tarjetas del intervalo que el usuario indique (por defecto se muestran todas las fechas disponibles). Este filtro comprueba que se haya seleccionado una fecha válida (que la fecha de inicio no sea mayor que la de fin) y agrupa y muestra solo los ficheros dentro de ese rango.

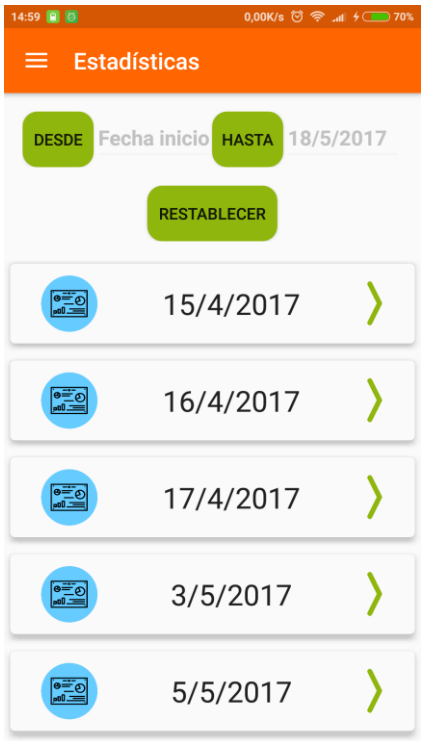

Figura 4-4: Vista principal de la sección de estadísticas

4.4 Estadísticas básicas

4.4.1 Tarjetas de eventos

Una vez se ha seleccionado una fecha de la que mostrar las estadísticas, se accede directamente a la vista básica de las mismas, donde podemos ver de cada evento los parámetros indicados en el RF4, como se muestra en la figura 4-5. Como puede verse, se trata de una vista sencilla con la información justa y necesaria, con iconos que ayudan a la interpretación de los datos y la navegación.

Figura 4-5: Ejemplo de la vista básica de estadísticas

Para la obtención de los datos de la tarjeta, desde la selección de fecha llegará la ruta del pulsómetro y la ruta del fichero de eventos del día seleccionado. Mediante la ruta del fichero de eventos, se procederá a la lectura del mismo, buscando el indicador de "EVALUACIÓN_POSITIVA", que marcará el inicio del evento y las pulsaciones iniciales. Para hallar las pulsaciones finales con la mayor exactitud posible, se utiliza el fichero de pulsaciones, en el que se busca la medida más próxima al *timestamp* indicado en el fin de la regulación, y se recogen las pulsaciones de ese momento. El título de la regulación también se recoge del fichero de eventos.

Al igual que en la vista de fechas, las tarjetas de esta vista usan un *RecyclerView* para agilizar el proceso de carga al desplazarnos por la pantalla, pero, como a lo largo de un mismo día puede haber un número muy elevado de eventos, la carga inicial de los mismos puede llevar algún tiempo, por lo que se han utilizado los métodos de la clase *AsyncTask* de manipulación de hilos para mostrar un mensaje de carga en el hilo principal, mediante un *ProgressDialog.* mientras se cargan los datos en un hilo secundario.

Para cumplir con el RF3 de acceso a la regulación indicada, cada tarjeta guarda el id de la regulación a la que hace referencia, y así, en el momento que se pulsa, se accede a la base de datos de la aplicación para comprobar si existe tal regulación, caso en el que se llamará a la actividad de edición de las regulaciones, o si ya no existe, mostrará un mensaje de aviso indicando la ausencia de la regulación.

Desde la vista básica, se puede acceder a la vista avanzada de ese día, tal y como indica el RF1, pulsando el botón que aparece arriba a la derecha (figura 4-5). En el momento de pulsarlo, se enviarán las mismas rutas de ficheros que habían llegado desde la selección de fecha y, además, una lista con todos los *timestamp* iniciales de los eventos del día.

4.5 Estadísticas avanzadas

Para el desarrollo de estas gráficas se ha utilizado la librería mencionada en la sección de Estado del arte de este documento, GraphView, de la que se detallará a continuación su funcionamiento. Como ya se ha explicado en el apartado de Diseño, al entrar en la vista avanzada se carga la gráfica del pulsómetro por defecto, y a partir de esta se pueden seleccionar las demás, por lo que empezaremos explicando la carga de la gráfica de un solo eje del pulsómetro.

Al igual que para cargar los datos de la vista básica, para la vista avanzada se ha usado *AsyncTask*, puesto que la lectura de ficheros y la representación de todos los datos lleva bastante carga de trabajo, y de esta forma se informa al usuario de que se está llevando a cabo la carga mediante un mensaje en primer plano, mientras se ejecuta en segundo plano.

La actividad de representación de la gráfica recibe una serie de nombres para indicar qué se está representando, la ruta del fichero que tiene que representar, los ejes que se han solicitado en caso de que los tenga y la lista de los tiempos iniciales de eventos que se mencionó previamente.

El primer paso consiste en determinar si se trata de uno de los ficheros estándar, es decir, pulsómetro, giroscopio, acelerómetro o detector de pasos, o si, por el contrario, es otro fichero alternativo. Esto nos sirve para fijar mejor los títulos que aparecen en la gráfica.

A continuación, se procede a la lectura del fichero a representar, almacenando los datos en un array. Nos dimos cuenta de que representar todos y cada uno de los puntos que aparecen en el fichero era contraproducente, pues el usuario no necesita tantos datos en ningún caso, sino que lo que necesita es una vista que resuma lo que ha sucedido y la evolución que ha habido de una forma legible, por lo que decidimos establecer un límite de datos que, después de unas cuantas pruebas y ensayos, quedó fijado en 3000 datos por gráfica, como máximo. Por ello, tras leer el fichero para convertir las líneas leídas a puntos de datos, se muestrea, omitiendo un número determinado de puntos para no sobrepasar el límite de datos establecido.

La librería elegida guarda los puntos a representar en un tipo de dato llamado “DataPoint”, compuesto por un parámetro X y un parámetro Y. Para guardar nuestros datos en este formato necesitamos saber cuántos van a ser, y por eso, una vez realizado el muestreo de datos y dependiendo del número de ejes del fichero se inicializan las variables que los almacenarán con el tamaño adecuado (tamaño *div* en la figura 4-6).

```
ArrayList<DataPoint[]> dataPointEjes = new ArrayList<>();
DataPoint[] points = null;

if (ejes != null) {
    for (int eje = 0; eje < ejes.size(); eje++) {
        DataPoint[] pointsEje = new DataPoint[div];
        dataPointEjes.add(pointsEje);
    }
} else {
    points = new DataPoint[div];
}
```

Figura 4-6: Ejemplo de inicialización del tamaño de los DataPoints de un fichero con ejes y sin ejes

Una vez inicializado el tamaño, se procede a la inserción de los datos, para lo cual cogeremos el *timestamp* como valor del eje X y los datos del eje o ejes (las otras columnas del fichero) como valores para el eje Y.

```
float Y = Float.parseFloat(words[2]);
points[i] = new DataPoint(date1, Y);
```

Figura 4-7: Ejemplo de creación DataPoints

En la figura 4-7 se ve la inserción en el caso de una gráfica de un solo eje, donde *date1* sería el *timestamp* recogido del fichero parseado a tipo long, e *Y* sería el dato de la tercera columna del fichero, es decir, el único eje.

Una vez que tenemos todos los datos que se van a representar en formato DataPoint, se introducen en una serie del gráfico, en este caso, una serie de tipo "LineGraphSeries<DataPoint>", como se puede ver en la figura 4-8 como ejemplo para una gráfica de un solo eje.

```
LineGraphSeries<DataPoint> series = null;
series = new LineGraphSeries<>(points);
```

Figura 4-8: Ejemplo de creación de una serie

Después, se continúa con el establecimiento del valor inicial del eje X, empezando en el primer valor de tiempo que se encuentre en los datos, y acabando en la mitad del valor total, para así poder hacer zoom y scroll desde una posición inicial cómoda. Para ello, se usan las siguientes funciones de la librería, tras encontrar los valores deseados, "setMinX" y "setMaxX":

```
graph.getViewport().setMinX(setMinX);
graph.getViewport().setMaxX(setMaxX);
graph.getViewport().setXAxisBoundsManual(true);
```

Figura 4-9: Establecimiento de los valores del eje X de la vista inicial de la gráfica

La búsqueda del valor máximo y mínimo del eje Y es algo más compleja si la gráfica posee más de un eje, pues hay que encontrar el valor máximo y mínimo entre todos ellos. Una vez encontrado se usan funciones similares a las del eje X:

```
graph.getViewport().setYAxisBoundsManual(true);
graph.getViewport().setMinY(minY);
graph.getViewport().setMaxY(maxY);
```

Figura 4-10: Establecimiento de los valores del eje Y de la vista inicial de la gráfica

Para fijar los puntos de crisis en la gráfica, necesitamos situarlos en puntos muestreados que ya existan, ya que, si simplemente los dibujásemos en el instante exacto en el que suceden podrían quedar fuera de la gráfica y no tendrían sentido visual. Por lo tanto, aprovechando la lista de tiempos iniciales de los eventos que llega como parámetro, buscamos en los datos que ya hemos muestreado el que tenga el tiempo que mejor se ajuste al inicio del evento, utilizando su X como coordenada X del punto de crisis que se marcará en el gráfico, mientras que la coordenada Y del punto de crisis será: 0 en el caso de que el

fichero de datos seleccionado posea más de un eje, o la misma coordenada Y que tenga el punto en caso de que cuente con un solo eje.

Estos puntos se irán almacenando en otra variable de tipo “DataPoint” que, una vez completa con todos los puntos de crisis, se añadirá a una serie del gráfico de tipo “PointsGraphSeries” a la que se le dará una forma triangular, de color rojo, con un tamaño adecuado y el título correspondiente (dependiendo del idioma), como se muestra en la figura 4-11.

```
PointsGraphSeries<DataPoint> bgSerie;

bgSerie = new PointsGraphSeries<>(crisisPoints);

bgSerie.setShape(PointsGraphSeries.Shape.TRIANGLE);
bgSerie.setColor(Color.RED);
bgSerie.setSize(15);
bgSerie.setTitle("Punto de crisis");
```

Figura 4-11: Definición de la serie de puntos de crisis de la gráfica

Una vez que hemos definido todas las series del gráfico, podemos añadirlas al mismo:

```
// Serie principal
graph.addSeries(series);

// Serie de ptos. crisis
graph.addSeries(bgSerie);
```

Figura 4-12: Adición de las series a la gráfica

Así como establecer detalles del gráfico tales como título, etiqueta del eje Y, formato de la fecha del eje X y estilo de la leyenda:

```
graph.setTitleTextSize(65);
graph.getGridLabelRenderer().setVerticalAxisTitleColor(Color.BLUE);

graph.setTitle(titulo + " " + "del" + " " + fecha);
graph.getGridLabelRenderer().setVerticalAxisTitle(unidades);

// formato de la etiqueta de la fecha
DateFormat miDateFormat = android.text.format.DateFormat.getTimeFormat(graph.getContext());
graph.getGridLabelRenderer().setLabelFormatter(new DateAsXAxisLabelFormatter(graph.getContext(), miDateFormat));
graph.getGridLabelRenderer().setNumHorizontalLabels(mNumLabels);

// estilo de la leyenda
graph.getLegendRenderer().setVisible(true);
graph.getLegendRenderer().setAlign(LegendRenderer.LegendAlign.TOP);
graph.getLegendRenderer().setMargin(30);
```

Figura 4-13: Detalles del estilo de la gráfica

Y activar el desplazamiento de la gráfica (“scroll”) y el “zoom”:

```
// para el zoom de la grafica
graph.getViewport().setScalable(true);
// para el desplazamiento de la grafica
graph.getViewport().setScrollable(true);
```

Figura 4-14: Activar desplazamiento y zoom de la gráfica

Además de estos detalles, también se ha añadido un “listener” para que, al pulsar en un punto de la gráfica, aparezca un mensaje con las coordenadas del eje Y que corresponden a ese lugar.

Por último, se han elegido 5 colores distintos que combinan bien visualmente y que son fácilmente distinguibles unos de otros, para asignárselos a los distintos ejes en las gráficas con más de un eje. En el caso de que la gráfica tuviese más de 5 ejes (algo inusual), estos colores se repetirían cíclicamente.

A continuación, se muestra un ejemplo de cómo resulta la gráfica avanzada al elegir mostrar los 3 ejes del acelerómetro:

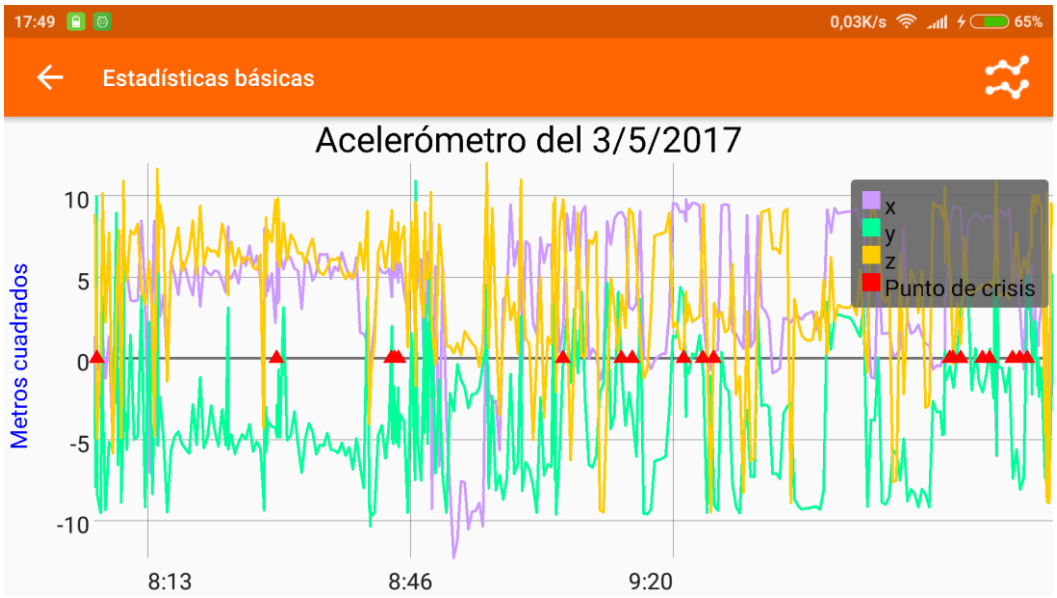

Figura 4-15 : Ejemplo de la vista avanzada de los datos del acelerómetro mostrando 3 ejes

5 Integración, pruebas y resultados

5.1 Pruebas unitarias

Las pruebas unitarias permiten analizar las distintas partes de la aplicación por separado para, en caso de encontrar un problema o error, saber qué falla exactamente.

En nuestro caso, para probar el correcto funcionamiento de la librería de representación gráfica GraphView, se creó una aplicación únicamente con el código de representación de los datos, y se crearon una serie de puntos aleatorios para incluirlos en la gráfica y ver el comportamiento de la misma, primero con puntos con coordenadas de tipo double, y luego añadiendo al eje X datos en formato fecha (Date).

Poco a poco, se fue aumentando el número de datos a representar, hasta que llegamos a un límite en el que el procesamiento de la gráfica comenzaba a ralentizarse, que fue aproximadamente a los 5000 puntos.

5.2 Pruebas de integración

Como el objetivo de este proyecto es formar parte de una aplicación ya desarrollada como lo es Taimun-Watch, tras el desarrollo por separado del mismo, hay que añadirlo como una sección más de la aplicación, por lo que se procedió a la adición de la sección de "Estadísticas" en el menú general de la aplicación y se comprobó el correcto funcionamiento de todas las posibles acciones que se pueden llevar a cabo en esta nueva sección, además de comprobar que el resto de la aplicación seguía funcionando con normalidad.

Como en la sección de las estadísticas básicas se puede acceder a la edición de una regulación, como ya se explicó en el punto 4.4, y esto pertenece a la otra parte de la aplicación, es importante realizar la comprobación de la correcta integración de esta funcionalidad, por lo que también se realizaron las pruebas necesarias para asegurar el correcto funcionamiento de este elemento de conexión entre eventos y regulaciones.

5.3 Pruebas funcionales

Estas pruebas aseguran el correcto funcionamiento de la aplicación, siguiendo con lo establecido por los requisitos funcionales. A continuación, se detallan las pruebas realizadas para la comprobación de los requisitos más susceptibles a error, una vez comprobados que todos los requisitos se cumplen para un caso de uso correcto, y otras pruebas adicionales que no tienen que ver con un requisito en concreto:

Prueba del RF2

- Descripción: se introduce una fecha inicial mayor a la fecha final del filtro de fechas.
- Resultado esperado: no se muestra ninguna tarjeta de fecha, pues ninguna coincide con la acotación especificada.
- Resultado de la prueba: correcto.

Prueba del RF3

- Descripción: se intenta acceder a una regulación que ya no existe porque ha sido eliminada.
- Resultado esperado: no se puede acceder a la edición y se muestra un mensaje de error.
- Resultado de la prueba: correcto.

Pruebas adicionales

Prueba adicional 1

- Descripción: se accede a la sección de estadísticas sin ningún fichero de datos de estadísticas disponible en la aplicación.
- Resultado esperado: se muestra un mensaje indicando que no hay ficheros que mostrar.
- Resultado de la prueba: correcto.

Prueba adicional 2

- Descripción: hay ficheros de datos de sensores, pero no existe el fichero de eventos.
- Resultado esperado: al entrar en la vista básica del día se muestra un mensaje indicando que no hay eventos que mostrar para ese día.
- Resultado de la prueba: correcto

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este trabajo de fin de grado, se ha llevado a cabo el desarrollo de un sistema de visualización de la información que se puede obtener de los sensores de un reloj inteligente, a partir de la aplicación Taimun-Watch para dispositivos móviles Android.

Para llevar a cabo el desarrollo del proyecto, se han seguido las fases del ciclo de vida del mismo, realizando primero un estudio de las librerías de código abierto que actualmente pueden llevar a cabo la representación de los datos necesarios para el proyecto, eligiendo la mejor para nuestro caso, también llevando a cabo el planteamiento de varios diseños hasta decidir el diseño definitivo que mejor cumplía con los objetivos de sencillez y usabilidad, desarrollando la codificación y, por último, realizando diversas pruebas sobre el producto final.

La aplicación cumple, finalmente, con todos los requisitos funcionales y no funcionales propuestos, cumpliendo los objetivos iniciales y los que se han ido añadiendo durante el desarrollo.

A nivel personal, he podido llevar a cabo el desarrollo y la integración de una sección de una aplicación en la que han trabajado muchas más personas, acordando ideas comunes, metas globales y en resumen, participando como una más en él, lo cual me deja con un gran sentimiento de contribución, con mi participación, a un proyecto más grande, que tiene como objetivo llegar y ayudar al mayor número de personas con Trastornos del Espectro Autista, esperando que, gracias a mi trabajo, se pueda lograr entender mejor el porqué de algunos comportamientos que, de momento, muchas veces no son fáciles de comprender, analizando pautas y comportamientos de los usuarios diariamente.

6.2 Trabajo futuro

Como trabajo futuro, nos gustaría añadir la posibilidad de que, desde un mismo dispositivo móvil con la aplicación Taimun-Watch, se pudiesen controlar las estadísticas de varios relojes inteligentes, es decir, de varios usuarios con TEA, ya que, actualmente, los datos recogidos por el reloj de cada usuario se guardan en la ruta de la aplicación, con una carpeta intermedia con el id del reloj, diferente en cada caso, por lo que solo necesitaríamos poder elegir el reloj del cual se quieren observar las estadísticas, y buscar su correspondiente carpeta.

Por otro lado, se plantea hacer pruebas con otros profesores que no han participado en el diseño de la aplicación para evaluar la idoneidad y usabilidad final.

Referencias

[1] Torrado Vidal, J. C., Montoro, G., & Gómez, J. (2016). The Potential of Smartwatches for Emotional Self-regulation of People with Autism Spectrum Disorder.
[2] Ambient Intelligence Laboratory (AmILab): http://amilab.ii.uam.es/
[3] Instituto de Psico-Pediatría Quintero Lumbreras, alenta: http://www.alenta.org/
[4] Androidplot [Última visita 3/5/2017]: https://github.com/halfhp/androidplot
[5] WilliamChart [Última visita 3/5/2017]: https://github.com/diogobernardino/WilliamChart
[6] MPAndroidChart [Última visita 3/5/2017]: https://github.com/PhilJay/MPAndroidChart
[7] GraphView [Última visita 3/5/2017]: http://www.android-graphview.org/
[8] Transparencias asignatura de Ingeniería del Software "Metodología, Ciclos de vida y Procesos software", Unidad 2, 55-56, curso 2015-2016.
[9] Imagen del ciclo de vida iterativo: http://sings-ufps.blogspot.com.es/search/label/Ingenier%C3%ADa%20de%20Software

Glosario

API	Application Programming Interface
Smartwatch	Reloj inteligente
Dispositivo Weareable	Dispositivo ponible, tecnología ponible
Timestamp	Secuencia de caracteres que representa una fecha y hora específicas
Listener	Objeto escuchador de eventos en la interfaz